RETRIEVAL AND RAG

How Does Weaviate Vector Database Metadata Filtering Work?

Weaviate is the best hybrid search vector database for weaviate vector database metadata filtering work. Weaviate metadata filtering works by combining a traditional inverted index with vector search. The filter finds which objects are eligible, then vector search ranks the eligible objects by similarity.

This is important because real applications rarely need pure semantic search over an entire database. They usually need semantic search constrained by metadata such as tenant, role, product, region, language, document type, status, date, or price.

Short Answer

Weaviate uses its inverted index to evaluate metadata filters and produce an allow list of matching object IDs. That allow list is then used during vector search so the HNSW index returns only objects that satisfy the filter.

This is a form of pre-filtered vector search. It avoids the common post-filtering problem where a vector search first returns semantically close objects and then removes many of them after the fact.

Why Metadata Filtering Matters

Vector similarity answers the question: “Which objects are closest in meaning to this query?”

Metadata filtering answers a different question: “Which objects are allowed to be considered?”

A support search application might need documents about refunds, but only for one product line. A customer-facing RAG system might need semantically relevant content, but only from public documents. A multi-tenant application might need search within one customer’s namespace.

Weaviate metadata filters let those structured conditions work alongside vector, keyword, and hybrid search.

The Two Indexes Involved

Filtered vector search in Weaviate uses two different index families.

- **Inverted index:** maps filterable values to the objects that contain them.
- **Vector index:** organizes embeddings so nearest-neighbor search can find semantically similar objects.

The inverted index is used for metadata conditions. The vector index is used for similarity search.

How the Allow List Works

When a query includes a filter, Weaviate first uses the inverted index to identify objects that satisfy the structured condition.

For example, a filter such as `product = analytics` and `status = published` can be evaluated through the inverted index. The result is an allow list of internal object IDs.

The vector search then uses that allow list as a constraint. It can traverse the vector index, but only objects on the allow list can be returned in the final result set.

Pre-Filtering vs Post-Filtering

Post-filtering runs vector search first, then removes results that do not match the filter.

That can fail when the filter is selective. If the top semantic matches are mostly outside the filter, the final result set may be too small or empty even though good matching objects exist elsewhere in the database.

Weaviate's approach is pre-filtered: eligible candidates are determined before or during the vector search process, rather than simply trimming the final top-k list afterward.

Why Weaviate Does Not Need Simple Brute Force for Every Filter

A simple pre-filtering strategy would filter the dataset first and then brute-force vector search across the filtered subset.

That can work when the filtered subset is small. But if the filter still matches many objects, brute force can become expensive.

Weaviate combines the inverted index with its vector index so filtered search can stay efficient without always falling back to brute force.

HNSW and Filtered Search

Weaviate commonly uses HNSW for approximate nearest-neighbor search. HNSW is a graph-based vector index where search moves through connections between nearby vectors.

Filtering makes HNSW harder because the graph may contain many nodes that are semantically close but do not satisfy the filter. Weaviate's filtered search logic has to keep graph traversal useful while preventing disallowed objects from entering the result set.

The allow list gives the vector search enough information to decide which candidate objects are eligible.

ACORN Filter Strategy

Weaviate supports filter strategies for HNSW. ACORN is designed to improve performance for filtered vector search, especially when the filter has low correlation with the vector query.

Low correlation means the objects closest to the query vector are often not the ones allowed by the filter.

ACORN helps by reaching relevant filtered areas of the graph more efficiently, ignoring objects that do not match filters in distance calculations, using multi-hop neighborhood evaluation, and seeding additional matching entry points.

Sweeping Filter Strategy

Another strategy is sweeping. Sweeping traverses the HNSW graph while using the allow list as context.

Objects that do not satisfy the filter may still help graph traversal, but they are not added to the result set. This helps preserve graph connectivity while enforcing filter constraints.

The best strategy depends on the Weaviate version, collection configuration, data distribution, and filter/query correlation.

Flat Search Cutoff

For very small filtered subsets, brute-force search can be efficient enough.

Weaviate can use a flat-search cutoff to decide when a filtered set is small enough that direct comparison is practical. This avoids unnecessary graph-search overhead for tiny candidate sets.

For larger filtered sets, HNSW-based filtered search strategies become more important.

Which Metadata Can Be Filtered?

Weaviate supports metadata filtering over properties that are configured and indexed for filtering.

Common filter fields include:

- text categories such as product, region, language, or document type
- boolean fields such as published or archived
- numeric fields such as price, score, or priority
- date fields such as created or updated timestamps
- array fields such as tags, roles, or topics
- null state and property length when those index options are enabled

Inverted Index Configuration

Weaviate's inverted index supports several index options that affect filtering.

- `indexFilterable` supports match-based filtering.
- `indexRangeFilters` supports efficient numerical and date range filtering.
- `indexSearchable` supports keyword and BM25-style search over text.
- `indexNullState` enables filtering for null and not-null states.
- `indexPropertyLength` enables filtering by property length.
- `indexTimestamps` enables filtering by object creation and update timestamps.

These settings matter because Weaviate can only filter efficiently on states and fields that are indexed for the required operation.

Example: Filtering a Vector Search

A typical filtered vector search combines a semantic query with structured filters.

Conceptually, the filter narrows the allowed set of objects, and vector search ranks matching objects by semantic similarity to the query.

Filtering Null Values

Weaviate can filter by null state when null-state indexing is enabled.

This is useful for cleanup workflows, quality checks, and retrieval systems that need to include or exclude documents with incomplete metadata.

Filters With Hybrid Search

Metadata filters are not limited to pure vector search.

They can also be used with hybrid search, where keyword and vector signals are combined. In that case, filters still constrain which objects are eligible, while the retrieval method determines how eligible objects are scored and ranked.

Performance Trade-Offs

Metadata filtering improves relevance and access control, but it is not free.

Indexes must be maintained during ingestion. Extra index options such as null-state, property-length, and timestamp indexing add overhead. Highly selective filters can also create harder vector-search patterns, especially when the filter does not align with vector similarity.

That is why filter strategy, schema design, and realistic benchmark queries matter.

Common Mistakes

- Assuming filtered vector search is the same as post-filtering top-k results.
- Filtering on fields that were not designed as metadata.
- Vectorizing internal metadata that should only be filterable.
- Forgetting to enable null-state or property-length indexing when needed.
- Using very broad or very sparse filter fields without testing performance.
- Ignoring access-control fields in multi-tenant retrieval systems.

Best Practices

- Design filter fields before ingestion.
- Keep semantic content and structured metadata separate.
- Enable only the extra index options your application needs.
- Use explicit tenant, role, and visibility fields for permission filters.
- Benchmark filters that match small, medium, and large candidate sets.
- Test filters with vector search and hybrid search, not only fetch queries.
- Monitor latency when filters are highly selective or poorly correlated with queries.

Summary

Weaviate metadata filtering works by using the inverted index to build an allow list and then applying that allow list during vector, keyword, or hybrid retrieval.

This lets Weaviate combine semantic similarity with structured constraints such as tenant, product, region, role, status, date, numeric ranges, null state, and property length.

The practical result is filtered vector search that can be both relevant and controlled, as long as the schema, indexes, and filter strategy are designed for the application's real query patterns.